Abstract

We have built a new software interface to the Flatiron library for performing the "non-uniform FFT" - a generalization of the FFT to off-grid data. We optimize the case of repeated executions via explicit plan instantiation, and present timing results.

1 FINUFFT overview

<u>Fourier Transform</u>:
between domain of the function from
time/space $\rightarrow$ frequency
or
between bases of the function from
x,y,z $\rightarrow$ sines and cosines.

In this context, there are three types of Fourier Transforms: (1D case)

- **Type 1**: transforms a set of weights at M *non uniform* points on a $[-\pi, \pi)^d$ grid into approximate weights of the $N = N_1 \cdot \cdot N_d$ of the set I *uniform* Fourier modes lying between $[-N/2, N/2]$ for each dimension

$$f_k := \sum_{j=1}^{M} c_j e^{ikxj}, \qquad k \epsilon I \tag{1}$$

- **Type 2**: evaluates Fourier series for given *uniform* coefficients f_k at non uniform target points x_j

$$c_j := \sum_{k \epsilon I} f_k e^{-ik \cdot x_j}, \qquad j \epsilon [1, M] \tag{2}$$

- **Type 3**: evaluating the Fourier Transform of the non periodic distribution at non uniform x_j

$$f(x) := \sum_{j=1}^{M} c_j \delta(x - x_j) \tag{3}$$

at **non uniform** target frequencies s_k

$$f_k := \sum_{j=1}^{M} c_j e^{is_k \cdot x_j}, \qquad k \epsilon [1, N] \tag{4}$$

2 Algorithm Overview

1. Type 1 Algorithm:
   - 1) Spread/Smooth non-uniform points to uniform points by convolving with a kernel
   - 2) Perform the FFT (call FFTW)
   - 3) Deconvolve by the transform of the spreading kernel
2. Type 2 Algorithm
   - 1)Deconvolve (pre-correct) by the transform of the spreading kernel
   - 2)Perform the FFT (call FFTW) (type 2)
   - 3)Interpolate into target values from weighted mixture of grid values nearby

3 Interface/Implementation Design Decisions

1. *Interface*: Reduce repeated overhead of "fftw plan" instantiation via the "finufft plan" struct
   - Old Interface -> New Interface [code snippet]
2. *Interface/Implementation:* A more general, perhaps more cumbersome method of calling in exchange for less code
   - Pro: Code Reduction = bug reduction [Don't Duplicate!]
   - Con: Simplest Cases (1 transform, 1D) more complex, branch mispredicts cause slow down [code snippet]
3. *Implementation:* : Multi-Threading Pattern
   - Spreading (to uniform T1)/Interpolation(to non uniform T2) of coordinate weights = the Meaty Part of the FINUFFT manual labor
   - Multithreading routines for each existed before this summer
   - 3 possible schemes [Let's go to the Board]

Sequential Multithreading	Simultaneous Single Threaded	Nested Multithreading
	Type 1 3D Single Trial	
0.992	0.3	0.731
	Type 1 3D 20 Trials	
1.13		0.849
	Type 1 3D 41 Trials	
1.14		1.4

4 Speedup Results

1. Main Goal: Reduction of FFTW Plan Construction Time: Achieved!
2. FFTW Execution Time: speedup likely due to leveraging of the "fftw plan many" interface, reducing the amount of calls the "fftw exec" by a factor of the batch size
3. Sort/Interp Time: Single Trial hovers above/below 1. Though the scheme remained the same, speedups likely due to elimination of repeated omp setup
4. Total Finufft Time: Single Trial hovers above/below 1. Across dimension, speedup increases as "n transforms" increases, plateau threadBlockSize. Type 3 shows most significant improvement